1

Submitted May, 1988 to the third annual WESTEX conference, sponsored by the Western Committee of the Computer Society of the IEEE and the IEEE Los Angeles Council.

Star-O/S - A Distributed Operating Environment for Cooperating Heterogeneous Systems

Dr. James R. Greenwood
Dr. H. Stephen Kaye

Advanced Decision Systems [*]
1500 Plymouth Street
Mountain View, CA 94043-1230

Abstract

The Star-O/S distributed operating environment was developed to provide communication and synchronization between multiple cooperating expert systems. Each expert system executes as a distinct logical process within the Star-O/S environment with access to multiple Common Lisp environments coordinated by the system.

Introduction

Star-O/S is a distributed operating environment developed for the AirLand Battle Management (ALBM) project supported by the DARPA and the U.S. Army*. This project is to develop a Corps and Division level distributed planning system consisting of several concurrent knowledge-based systems in a multiple workstation environment. Several distinct knowledge base tools are being utilized, including the Knowledge Engineering Environment (KEE) from Intellicorp, the Procedural Reasoning Systems (PRS) from Advanced Decision Systems, and the IRUS natural language system from Bolt Beranek and Newman (BBN). Star-O/S provides the "clue" to tie these systems together at both the system/environment level, and also at the language level (i.e., Common Lisp and C).

Like several other systems, most notably the V-system from Stanford University, and the Hetergeneous Computer System (HSC) at the University of Washington, Star-O/S assumes a workstation environment distributed on a local area network (LAN). Like those systems, it assumes a high-speed reliable LAN, perhaps hetergeneous processors, and a variety of applications. Star-O/S also assumes hetergeneous underlying operating systems, and direct interfaces into multiple single-user Common Lisp environments.

Other similar distributed operating systems are targeted to general purpose time-share computing, with emphasis on mail systems, distributed file systems, and remote

* This research was supported by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Ballistic Research Laboratory (BRL) under subcontract 7A40B0820M from Lockheed Missiles & Space Company, Inc., Austin Division.

computation. Star-O/S is tailored to embedded application execution environments where multiple processes are loaded on several processors and concurrently perform as a single application. This is particularly important when dealing with Common Lisp environments typically used to develop expert systems. The Common Lisp environments, in general, are designed as single-user interactive development environments. As such, they universally assume a "user/developer" is always "sitting" at the console. In a complex application, such as ALBM, this is not the case; each and every Common Lisp execution environment must be "slaved" to the overall application control environment.

In order to allow this, each Common Lisp environment is accessible across the network in either a synchronous or asynchronous manner such that each executing expert system can coordinate with the others. An important aspect of this coordination is error recovery, whereas an error in one environment caused by an activity in another environment is detected and "caught" and then passed back to the "causing" agent. Thus, Star-O/S provides a set of facilities that allow multiple "single-user" Common Lisp environments to work as a single application with cooperating agents.

Star-O/S also provides a simple interface into other processes written in other languages, such as the C language. Several of the complex graphics aspects of the ALBM system are written in C and integrated via the Star-O/S mechanisms.

In order to quickly provide a set of capabilities for specific projects, our design of Star-O/S makes maximal use of existing technology and standards, such as Network File Service (NFS), the UDP/IP protocol, and standard Common Lisp environments on Symbolics, Sun, and Silicon Graphics hardware. This approach is discussed below relative to alternative systems.

Distributed Operating Systems Approaches

Operating systems in general must satisfy two complementary, yet sometimes conflicting, needs. The first is to manage resources, both hardware (e.g., processors, disks, devices) and software (e.g., compilers, data base systems) of a computer system. The second is to provide services that utilize the hardware and software resources to support specific applications. Key aspects in delivering services in an operating system are the notion of process, the communication between processes, and the synchronization (both data and control) between processes. Inherent in the development of distributing operating systems is the fact that services and resources are distributed across a set of possibly hetergeneous processors.

Several different approaches to distributed operating systems are utilized, with a primary distinction being **kernel** versus **layered** systems. A kernel distributed O/S provides the foundation "kernel" of the operating system on each machine; that is, it replaces any other operating system. For example, the MACH operating system developed at Carnegie Mellon University is an example of a "kernel" system. A layered distributed O/S surrounds an existing operating system (e.g., UNIX, Genera) with a shell such that each application sees a consistent set of services and resources on any machine regardless of operating system. Both the V-system and the HSC system mentioned earlier are examples of "layered" systems. A layered distributed O/S is easier to implement, but typically less efficient, than a kernel distributed O/S.

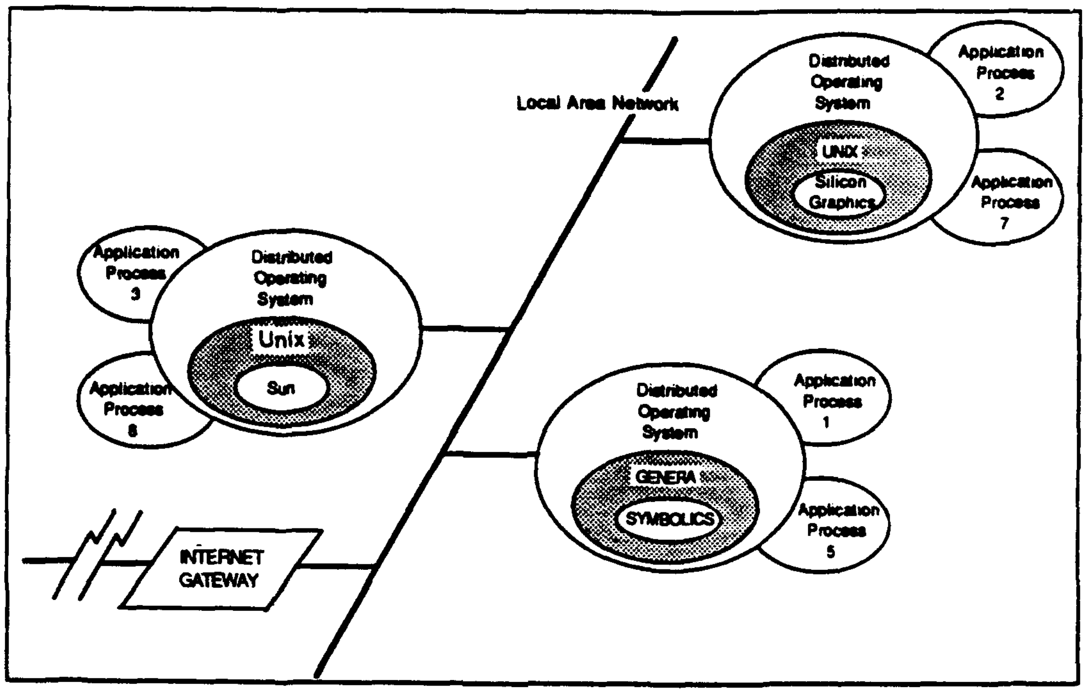


Figure 1: Star-O/S is a layered Distributed Operating System

Figure 1 shows a layered distributed operating system of Star-O/S approach. Each application sees a consistent operating system interface even though hosted in a hetergeneous environment. Star-0/S synchronizes and provides communication between application processes over the network.

Client-Server

A key aspect of most distributed operating systems is the **client-server** relationship between processes. At any one time a process is either a client or server. Most operating system services can be viewed as server operations. Client-server relationships allow multiple processes to cooperate in performing a single application. Star-O/S utilizes the client-server relationship between separate Common Lisp environments. During processing, at any one time a process is either a client or server.

Star-O/S is built upon the Remote Procedure Call (RPC) mechanism supplied in various network systems, such as Sun's Network File System (NFS) and ILA's NFS package for the Symbolics Lisp Machine. The Remote Procedure Call (RPC) looks like a local procedure call, but allows a client process on one machine access to data, knowledge bases, routines, and devices on another machine, as shown in Figure 2. An RPC causes a daemon process running on the target machine to execute some software on the client's behalf. The RPC is the fundamental building block upon which the higher level functionality of Star-O/S communication is layered as shown in the next figure.

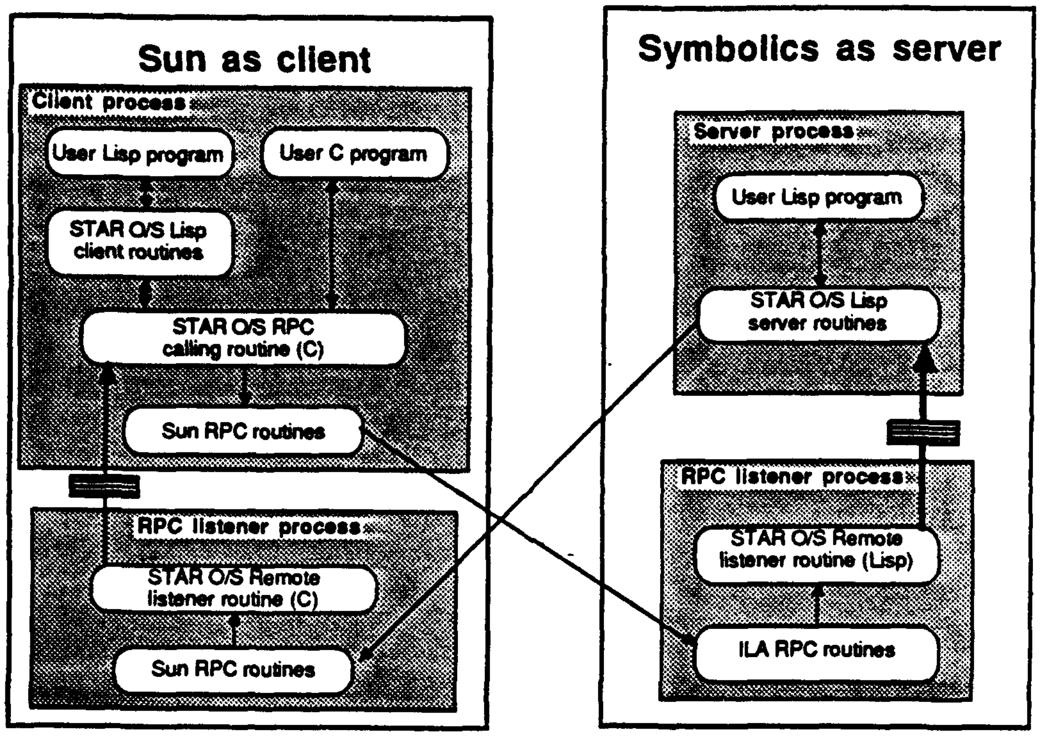


Figure 2: Common Lisp client-server data flow in Star-O/S

Figure 2 shows message flow between various clients and servers on the two machines. Each shaded block is a separate process on the respective machine. Processes on the same machine utilize either UNIX sockets, and/or shared data areas for communication, while communications between processes utilize the RPC mechanism. Message acknowledgment, time-out, and buffering are supported in the communication. The striped box in the message flow between processes represents a FIFO queue.

The flow of information as shown can be understood by following a message through the system. For instance, in Figure 2 a client on the Sun from Common Lisp invokes a "star-os:eval-on" which is a Star-O/S Lisp client routine. It asks the Star-O/S routines that perform logical name translation to determine the destination (see following section). The message is sent via the Sun RPC routines to the ILA RPC routines in the Listener process on the designated Symbolics processor. The message is passed to the Star-O/S routines within the Listener process which determine the actual process on the Symbolics to which the message is destined. The Listener process queues the message for the Symbolics resident server process. The Star-O/S routines in the server process pick up the message and make it available for the user's server process. The Server evaluates the lisp expression sent and sends back the reply, which goes through a similar sequence to return to the client. The client receives the evaluation of the expression.

Providing cooperating agents on multiple machines requires distributed O/S process synchronization, coordination, as well as error handling. Of particular importance is error handling, so that an error caused by a process A within another process B is caught by the Star-O/S and passed back to process A for handling in process A. Star-O/S is integrated into each Common Lisp environment to "catch" errors in the server environment and pass them back to the client Common Lisp environment. Thus, a server does not enter the Lisp debugger on an error caused by a client process. Star-O/S currently supports serveral Common Lisp environments, including Symbolics Common Lisp and both Franz and Lucid Common Lisp on the Sun workstations.

Communication Between Client and Server

The communication between processes can be accomplished with varying degrees of synchronization depending on the level of the local context, and the programming language employed. At a single process level (e.g., C programs on each machine), two types of client/server communication are evident: synchronous and asynchronous without reply, shown in figure 3 below:

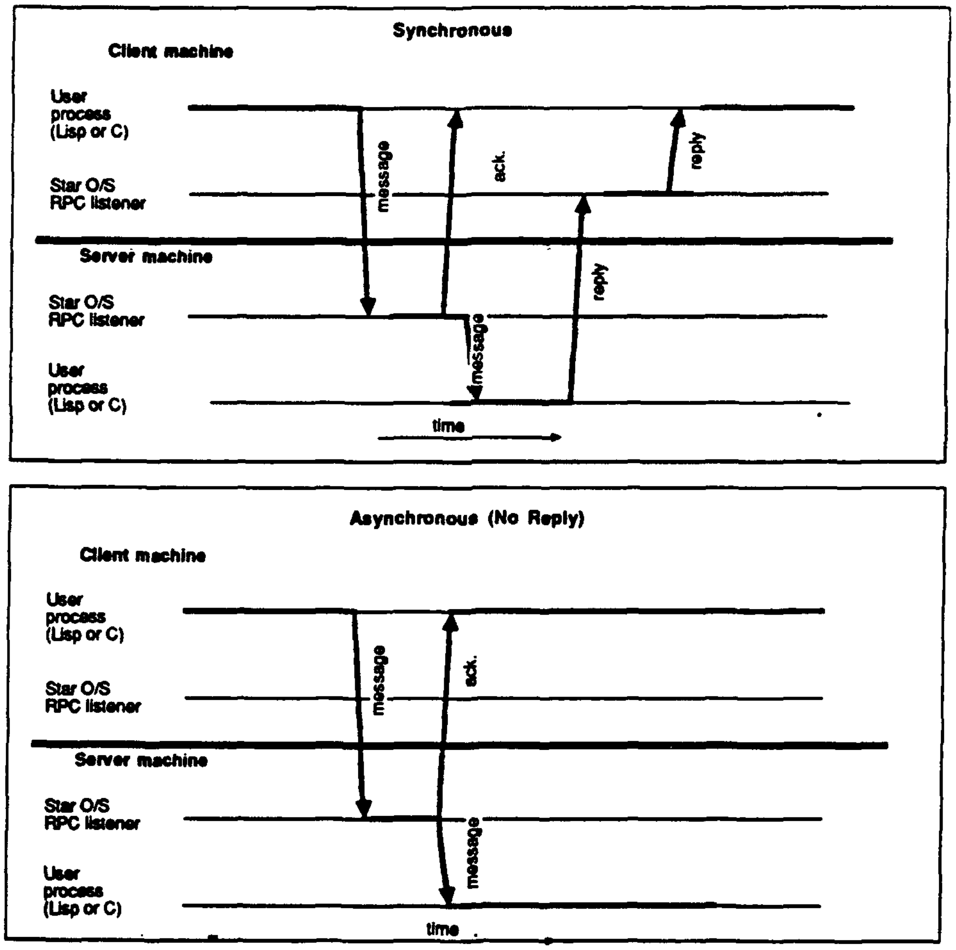
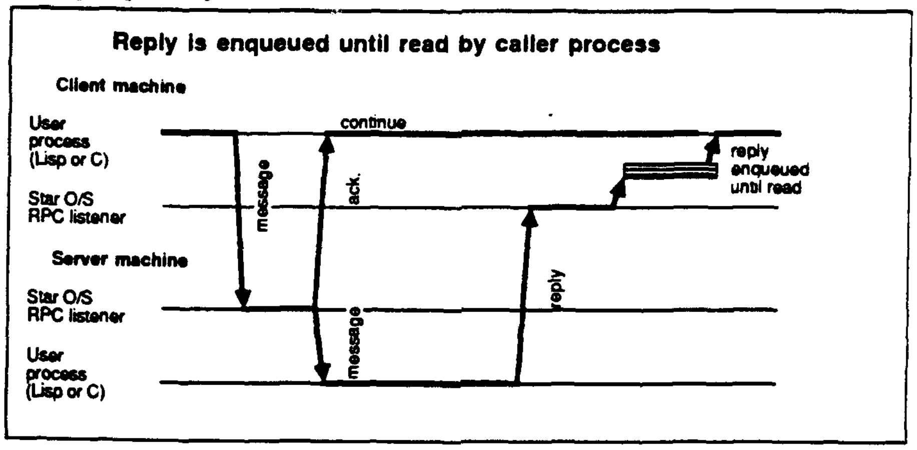


Figure 3: Client/Server Communication in Star-O/S

The synchronous communication is essential for closely coupled processing where the client process must wait for the server process to complete before proceeding. For instance, on the ALBM project the interaction between the maneuver planning system (i.e., MOVES) is synchronous with the war-gaming system used to evaluate a plan. On the otherhand, the ALBM subsystem for the Fire Support (i.e., FIRES) can be tasked by the overall system and run concurrently with MOVES. The asynchronous communication of Star-O/S is used for that concurrent control.

In systems that incorporate some sort of context scheduler within a process (e.g., object-oriented systems such as SOPE of CLOS), an additional client/server communication is possible: asynchronous with reply (See Figure 4). In this form, the client can continue processing while the server performs the request. The client then processes the reply when it is received. This form of communication gives the concurrent processing advantages of the previous asynchronous communication, but with the control and synchronization like synchronous communication. However, this form of communication requires a multi-stream processing capability in the client environment.



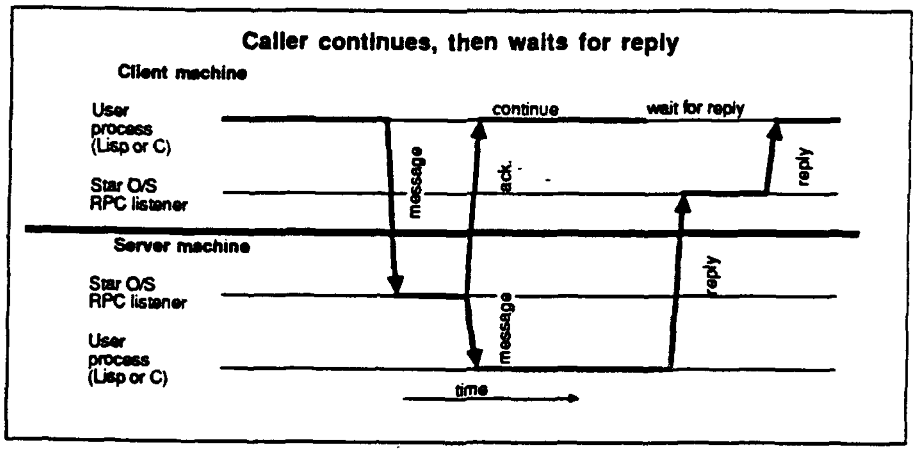


Figure 4: Asynchronous Communication with Reply

Logical Processes and Context

In order to facilitate machine and operating system independence, Star-OS provides a "logical name" facility that is used to declare names by which processes, and contexts within processes, are identified in Star-O/S messages. All Star-O/S functionality is addressed via these logical names. Logical names can be attached to the physical processes, and nested to redirect communication when processes are configured on different machines or at different times. This provides machine independence and easy reconfiguration of a system composed of multiple processes.

Logical names are stored in a single global name space and can be defined and redefined by any application at run-time. Processes communicate and synchronize by utilizing the current mappings of the logical names. Subprocesses, such as objects in object-oriented systems, can also be assigned logical names, thus allowing a client process to specify a context in which its message is to be evaluated.

Thus, while communication between processes is point-to-point (for efficiency) in a client/server relationship, the precise physical processes involved depend on the current state of the distributed logical name space. Logical names are recorded within individual shared mapping tables in each processor. This table is shared between processes on each processor for rapid access and copies of the tables are distributed to all machines. The Star-O/S Listener maintains the consistency of the tables when logical names for processes and subprocesses are created and changed.

Future Directions and Summary

Star-O/S is currently being extended to support higher level cooperation mechanisms including global process events and system-wide process control. Processes on any machine will be able to start, stop (pause), abort, and extract any other logical process. Any process can conditionally wait or synchronize with a global event. Global events can be triggered by processes on any machine. These features will allow application-dependent process schedulers to be written with Star-O/S primitives on heterogeneous machine architectures, and provide further cooperating system mechanisms.

Star-O/S has proven quite successful in use on the ALBM Project for which it was developed. It supplies the foundation for the cooperating knowledge-based systems written in Common Lisp that comprise ALBM project. Star-O/S has shown that a layered distributed operating system is a viable solution for problems that require synchronization, information sharing, and flexible task distribution among several processes on several, possibly heterogeneous, machines.

References

1. Almes, G. The impact of language and system on remote procedure call design. In *Proceedings of the 6th International Conference on Distributed Computer Systems* (Cambridge, Mass., May 19-23). Computer Society, Los Angeles, Calif., 1986, pp.414-421.
2. Archibald, J., and Baer, J.L. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Trans. Comput. Sys.* 4,4 (Nov 1986) pp 273-298.
3. Baskett, F., Howard, J.H, and Montague, J.T. Task communications in DEMOS. In *Proceedings of the 6th Symposium on Operating System Principles* (Perdue University, W. Lafayette, Ind., Nov 16-18, 1977). ACM, New York, 1977, pp. 23-31.
4. Brinch Hansen, P. The nucleus of a multiprogramming system. *Commun. ACM* 13,4 (Apr 1970), 238-241, 250.
5. Cheriton, D.R., The V Distributed System. The *Commun. ACM* 31,3 (Mar 1988), 314-333.
6. Cheriton, D.R., Local networking and internetworking in the V-system. In *Proceedings of the 8th Symposium on Data Communication (*North Falmouth, Mass., Oct 3-6). IEEE/ACM, Los Angeles, Calif.,1983, pp. 9-16.
7. Cheriton., D.R. Problem-oriented shared memory: A decentralized approach to distributed system design. In *Proceedings of the 6th International Conference on Distributed Computer Systems* (Cambridge, Mass., May). IEEE Computer Society, Los Angeles, Calif., 1986, 190-197.
8. Cheriton., D.R. and Mann, T.P., Decentralizing: A global naming service for efficient fault tolerant access. *ACM Trans. Comput. Syst.* (1988) To appear. An earlier version

appeared as Tech. Rep. STAN-CS-86-1098, Computer Science Dept., Stanford University, April 1986.
9. Faulk, S.R., and Parnas, D.L. On synchronization in hard-real-time systems. In *Commun. ACM* 31,3 (mar 1988), 274-287.
10. Gosney, K. Hetergeneous remote procedure call for Franz Lisp. M.S. Thesis and Tech. Rep. 87-0703, Dept. of Computer Science, Univ. of Washington, Seattle, July 1987.
11. Hayes, R. , and Schlichting, R.D. Facilitating mixed-language programming in distributed systems. IEEE *Trans. Softw. Eng.* SE-13,12 (dec 1987), 1254-1264.
12. Lampson, B. Designing a global name service. In *Proceedings of the 5th Annual Symposium on Principles of Distributed Computing* (Calgary, Canada, Aug 11-13). ACM, New York, 1986, pp. 1-10.
13. Lantz, K.A., and Nowicki, W.I., Structured Graphics for distributed systems. ACM *Trans. Graph.* 3,1(Jan 1984), 23-51.
14. Liskov, B. Distributed Programming in Argus. In *Commun. ACM* 31,3 (Mar 1988), 300-312.
15. Notkin, D., Black, A.P., Lazowska, E.D., Levy, H.M., Sanslo, and J., Zahorjan, J. Interconnecting Hetergeneous Systems. In *Commun ACM* 31,3 (Mar 1988), 258-273.
16. Rashid, R. Mach: A new kernel foundation for Multiprocessor Unix development. Carnegie Mellon Unicversity (1988). To appear.
17. Ritchie, D.M., and Thompson, K. The UNIX timesharing system. *Commun. ACM* 17,7 (July 1974), 365-375.
18. Schwartz, M., Zahorjan,J., and Notkin, D. A name service for evolving hetergeneous systems. In *Proceedings of the 11th ACM Symposium on Operating System Principles* (Austin, Tex., Nov.). ACM, New York, 1987.
19. SUN Microsystems. Network File System Specification. Sun Microsystems, Mountain View, Calif. 1985.
20. SUN Microsystems. Remote Procedure Call Protocol Specification. Sun Microsystems, Mountain View, Calif., Jan. 1985.
21. Terry, D. Distributed name servers: Naming and caching in large distributed computing environments. Ph.D. dissertation, Computer Science Division (EECS), Univ. of California, Berkeley, Feb 1985.
22. Xerox Corporation. Courier: The remote procedure call protocol. Tech. Rep. XSIS 038112, Xerox Corporation, Dec. 1981.